

# Tomcat Web Application Security

NOTE: For this document, I borrow *heavily* from <http://tomcat.apache.org/tomcat-5.5-doc/realm-howto.html>.

## What is a Realm?

A Realm is a "database" of usernames and passwords that identify valid users of a web application (or set of web applications), plus an enumeration of the list of *roles* associated with each valid user. You can think of roles as similar to *groups* in Unix-like operating systems, because access to specific web application resources is granted to all users possessing a particular role (rather than enumerating the list of associated usernames). A particular user can have any number of roles associated with their username.

Several types of databases are supported. Examples include

Database	Description
JDBCRealm	Accesses authentication information stored in a relational database, accessed via a JDBC driver.
DataSourceRealm	Accesses authentication information stored in a relational database, accessed via a named JNDI JDBC DataSource.
JNDIRealm	Accesses authentication information stored in an LDAP based directory server, accessed via a JNDI provider.
MemoryRealm	Accesses authentication information stored in an in memory object collection, which is initialized from an XML document (conf/tomcat-users.xml).
JAASRealm	Accesses authentication information through the Java Authentication & Authorization Service (JAAS) framework.

It is also possible to write your own Realm implementation, and integrate it with Tomcat 5. Here we focus on using a JDBCRealm to secure the personnel application.

## Configuring a JDBCRealm

A JDBCRealm stores information in a relational database. To begin, we must define the tables related to security:

- There must be a table, referenced below as the *users* table, which contains one row for every valid user that this Realm should recognize.
- The *users* table must contain at least two columns (it may contain more if your existing applications required it):
  - Username to be recognized by Tomcat when the user logs in.
  - Password to be recognized by Tomcat when the user logs in. This value may in cleartext or digested. We'll ignore the digested option; however, if storing actual passwords in the database makes you nervous, you can simply store a digest of the password.
- There must be a table, referenced below as the *user roles* table, which contains one row for every valid role that is assigned to a particular user. It is legal for a user to have zero, one, or more than one valid role.
- The *user roles* table must contain at least two columns (it may contain more if your existing applications required it):
  - Username to be recognized by Tomcat (same value as is specified in the *users* table).
  - Role name of a valid role associated with this user.

In the personnel application, we have two tables:

users

login	password
mike	mikemike
jane	janenator

roles

login	role
mike	admin
mike	user
jane	user

Note that roles.login is a foreign key to users.login. In this example, mike has the role of both admin and user, while jane is just a user.

Now that we've created our credential database, we need to describe it to Realm in the application configuration file. For the personnel application, this file is located in tomcat\conf\Catalina\localhost\personnel.xml:

```
<Context path="/personnel" docBase="personnel" debug="0" reloadable="true" crossContext="true">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="localhost_personnel_log." suffix=".txt" timestamp="true" />
  <Realm className="org.apache.catalina.realm.JDBCRealm" debug="99" driverName="com.mysql.jdbc.Driver"
    connectionURL="jdbc:mysql://localhost/personnel" connectionName="root" connectionPassword="3335rocks"
    userTable="users" userNameCol="login" userCredCol="password" userRoleTable="roles" roleNameCol="role" />
</Context>
```

The section in bold specifies two things:

- how Realm connects to the database: driverName, connectionURL, connectionName (DB login ID), and connectionPassword
- the specific tables and columns containing the credentials: userTable, userNameCol, userCredCol, userRoleTable, roleNameCol.

## Configuring Application Security

Finally, we must specify 1) which pages need to be secured and 2) what roles can access them. For the personnel application, we do this in tomcat\webapps\personnel\WEB-INF\web.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Personnel</web-resource-name>
      <!-- Define the context-relative URL(s) to be protected -->
      <url-pattern>/secure/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
```

```
    <role-name>user</role-name>
  </auth-constraint>
</security-constraint>

<!-- Default login configuration uses form-based authentication -->
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Personnel</realm-name>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/loginerror.jsp</form-error-page>
  </form-login-config>
</login-config>

<security-role>
  <role-name>user</role-name>
</security-role>
</web-app>
```

Our security specification is divided into three sections:

### **<security-constraint>**

We begin by identifying the pages that must be secured by Tomcat with the `<url-pattern>`. The `<auth-constraint>` specifies the roles a user must have to access these pages. For the personnel applications, accessing resources in the `/secure` directory requires a role of user.

### **<login-config>**

Next, we identify the method by which we will get the user credentials. In the personnel application, we use the FORM method. When the user first attempts to access a resource requiring authorization, Tomcat will present the login page specified by `<form-login-page>`. Successful users are forwarded to the resource. Unsuccessful users are directed to the error page specified by `<form-error-page>`.

### **<security-role>**

Finally, `<security-role>` lists all referenced security roles.